Initialization

```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.spatial.distance import cosine
from community import community_louvain
```

```
df = pd.read_csv ('https://raw.githubusercontent.com/thatNitinVinayak/Inventory-Management-using-IoT/main/References/Reference%20Code/dataset/Items.c
```

```
df.head()
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559	Bose SoundSport Headphones	1	99.99	04-07-2019 22.30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04-12-2019 14.38	669 Spruce St, Los Angeles, CA 90001

```
pathforxlsxfile = 'https://github.com/thatNitinVinayak/Inventory-Management-using-IoT/blob/main/References/Reference%20Code/dataset/OnlineRetail.xlsx
dataset = pd.read_excel(pathforxlsxfile)
```

```
print('Dataset Dimensions : ', dataset.shape)
dataset.describe(include = 'all')
```

```
Dataset Dimensions :  (541909, 8)
<ipython-input-5-94a0e184a5b8>:2: FutureWarning: Treating datetime data as categorical rather than nume
  dataset.describe(include = 'all')
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
count	541909.0	541909	540455	541909.000000	541909	541909.000000	406829.000000	54
unique	25900.0	4070	4223	NaN	23260	NaN	NaN	
top	573585.0	85123A	WHITE HANGING HEART T-LIGHT HOLDER	NaN	2011-10-31 14:41:00	NaN	NaN	U King
freq	1114.0	2313	2369	NaN	1114	NaN	NaN	49
first	NaN	NaN	NaN	NaN	2010-12-01 08:26:00	NaN	NaN	
last	NaN	NaN	NaN	NaN	2011-12-09 12:50:00	NaN	NaN	
mean	NaN	NaN	NaN	9.552250	NaN	4.611114	15287.690570	
std	NaN	NaN	NaN	218.081158	NaN	96.759853	1713.600303	
min	NaN	NaN	NaN	-80995.000000	NaN	-11062.060000	12346.000000	
25%	NaN	NaN	NaN	1.000000	NaN	1.250000	13953.000000	
50%	NaN	NaN	NaN	3.000000	NaN	2.080000	15152.000000	

```
dataset_sample = dataset.iloc[:4000]
```

Data Preprocessing

```
# Delete Rows with no Customer ID (if there is such a case)
cleaned_retail = dataset_sample.loc[pd.isnull(dataset_sample.CustomerID) == False]

# Create a Lookup Table
item_lookup = cleaned_retail[['StockCode', 'Description']].drop_duplicates()
item_lookup['StockCode'] = item_lookup.StockCode.astype(str)

# Data Cleaning to Raw Data
cleaned_retail['CustomerID'] = cleaned_retail.CustomerID.astype(int)
cleaned_retail = cleaned_retail[['StockCode', 'Quantity', 'CustomerID']]
grouped_cleaned = cleaned_retail.groupby(['CustomerID', 'StockCode']).sum().reset_index()
print()
grouped_cleaned.Quantity.loc[grouped_cleaned.Quantity == 0] = 1
grouped_purchased = grouped_cleaned.query('Quantity > 0')
```

```
<ipython-input-7-6d2fc95d9cd0>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  cleaned_retail['CustomerID'] = cleaned_retail.CustomerID.astype(int)

/usr/local/lib/python3.8/dist-packages/pandas/core/indexing.py:1732: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

```python
# Count Number of Products and Number of Customers in the Reduced Dataset
products_num = len(grouped_purchased.StockCode.unique())
customers_num = len(grouped_purchased.CustomerID.unique())
print('Number of Customers in Dataset:', customers_num)
print('Number of Products in Dataset:', products_num)
```

```
Number of Customers in Dataset: 137
Number of Products in Dataset: 1131
```

Creation of Bipartite Graph

```python
# Turn Raw Data to Pivot ('ratings' Matrix)
ratings = grouped_purchased.pivot(index = 'CustomerID', columns = 'StockCode', values = 'Quantity').fillna(0).astype('int')
# Binarize the Ratings Matrix (indicate only if a customer has purchased a product or not)
ratings_binary = ratings.copy()
ratings_binary[ratings_binary != 0] = 1
```

Conversion to a Weighted Product Graph

```python
# Initialize Zeros Dataframe for Product Interactions
products_integer = np.zeros((products_num, products_num))

# Count how many times each Product Pair has been Purchased
print('Counting how many times each pair of products has been purchased...')
for i in range(products_num):
    for j in range(products_num):
        if i != j:
            df_ij = ratings_binary.iloc[:,[i,j]] # Create a Temporary Dataset 'df' with only i and j Products as Columns
            sum_ij = df_ij.sum(axis=1)
            pairings_ij = len(sum_ij[sum_ij == 2]) # if s1_ij == 2 it means that both Products were Purchased by the Same Customer
            products_integer[i,j] = pairings_ij
            products_integer[j,i] = pairings_ij
```

```
Counting how many times each pair of products has been purchased...
```

```python
# Count how many Customers have Purchased each Item
print('Counting how many times each individual product has been purchased...')
times_purchased = products_integer.sum(axis = 1)
```

```
Counting how many times each individual product has been purchased...
```

```python
print (times_purchased)
```

```
[30. 73. 27. ... 20. 63. 34.]
```

Building Product Matrix

```python
# Construct Final Weighted Matrix of Item Interactions
print('Building weighted product matrix...')
products_weighted = np.zeros((products_num, products_num))
for i in range(products_num):
    for j in range(products_num):
        if (times_purchased[i] + times_purchased[j]) != 0: # Make sure you do not Divide with Zero
            products_weighted[i,j] = (products_integer[i,j])/(times_purchased[i]+times_purchased[j])
```

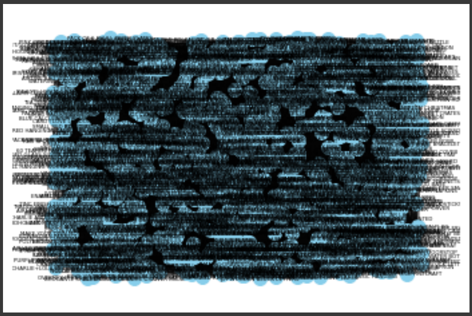
```
Building weighted product matrix...
```

Visualization of Weighted Product Matrix

```python
# Get List of Item Labels (instead of Codes)
nodes_codes = np.array(ratings_binary.columns).astype('str')
item_lookup_dict = pd.Series(item_lookup.Description.values,index = item_lookup.StockCode).to_dict()
nodes_labels = [item_lookup_dict[code] for code in nodes_codes]
```

```
# Create Graph Object using the Weighted Product Matrix as Adjacency Matrix
G = nx.from_numpy_array(products_weighted)
pos = nx.random_layout(G)
labels = {}
for idx, node in enumerate(G.nodes()):
    labels[node] = nodes_labels[idx]

nx.draw_networkx_nodes(G, pos, node_color = "skyblue", node_size = 100)
nx.draw_networkx_edges(G, pos,  edge_color = 'k', width = 0.3, alpha = 0.5)
nx.draw_networkx_labels(G, pos, labels, font_size = 4)
plt.axis('off')
plt.show()
```



Exporting the Graph to Gephi

```
# Create a New Graph with Description Labels and Save to Gephi for better Visualizations
H = nx.relabel_nodes(G, labels)
nx.write_gexf(H, "products.gexf")
```

Louvain Clustering

```
!pip install --upgrade scipy # upgrading Scipy to the Latest Version
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (1.10.1)
Requirement already satisfied: numpy<1.27.0,>=1.19.5 in /usr/local/lib/python3.8/dist-packages (from scipy) (1.22.4)
```

```
!pip install networkx==2.7 # downgrading Networkx to v.2.7
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: networkx==2.7 in /usr/local/lib/python3.8/dist-packages (2.7)
```

```
# Function for Setting Colors of Nodes and Edges
def get_paired_color_palette(size):
    palette = []
    for i in range(size * 2):
        palette.append(plt.cm.Paired(i))
    return palette

# Find Communities of Nodes (products)
louvain = community_louvain.best_partition(G, resolution = 1.5)
values = list(louvain.values())

communities = []

for i in set(louvain.values()):
    nodelist = [n for n in G.nodes if (louvain[n] == i)]
    communities.append(nodelist)
```

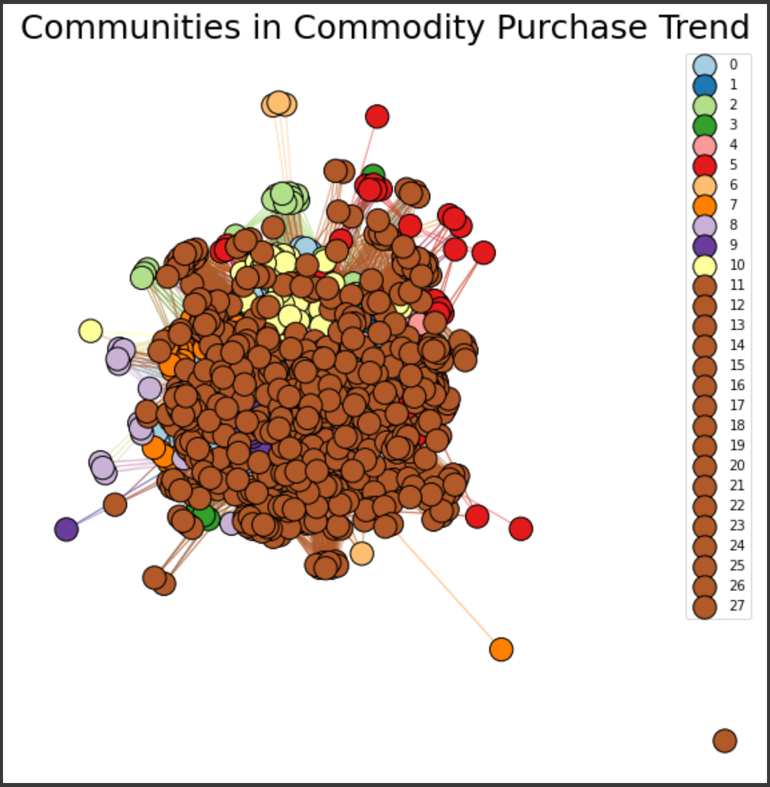
Visualize Detected Communities

```
# Make Plot using matplotlib, networkx spring_layout, set_colors using cluster_count and get_paired_color_pallette
clusters_count = len(set(louvain.values()))
plt.figure(figsize = (10, 10))
light_colors = get_paired_color_palette(clusters_count)
dark_colors = get_paired_color_palette(clusters_count)
g = nx.drawing.layout.spring_layout(G, weight = 'weight')

# Iterate through each of the Communities found by the Louvain Algorithm and Plot
for i in set(louvain.values()):
    nodelist = [n for n in G.nodes if (louvain[n] == i)]
    edgelist = [e for e in G.edges if ((louvain[e[0]] == i) or (louvain[e[1]] == i))]
    node_color = [light_colors[i] for _ in range(len(nodelist))]
    edge_color = [dark_colors[i] for _ in range(len(edgelist))]
    nx.draw_networkx_nodes(G, g, nodelist = nodelist, node_color = node_color, edgecolors = 'k', label = i)
    nx.draw_networkx_edges(G, g, edgelist = edgelist, alpha = .5, edge_color = edge_color)
```

```python

# Set Title, Legend and ShowPplot
plt.title('Communities in Commodity Purchase Trend', fontdict = {'fontsize': 25})
plt.legend()
plt.axis('off')
plt.show()
```

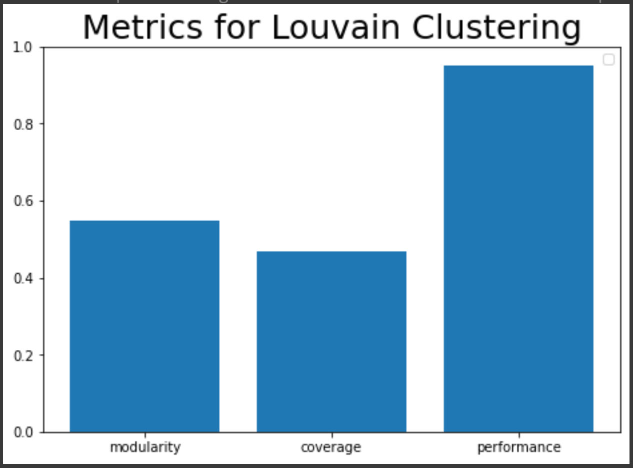


Performance Metrics

```python
# Transform all the Graphs to the Same Format
clusters = []
for cluster in range(len(set(louvain.values()))):
    cluster_list = []
    for k, v in louvain.items():
        if v == cluster:
            cluster_list.append(k)
    clusters.append(cluster_list)
louvain = clusters

x = ['modularity', 'coverage', 'performance']
y= [nx.community.modularity(G, eval('louvain')),
    nx.community.coverage(G, eval('louvain')),
    nx.community.performance(G, eval('louvain'))]
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.bar(x,y)
plt.title('Metrics for Louvain Clustering', fontdict = {'fontsize': 25})
plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label
```



Detected Community Analysis

```
# Check how many Communities were Created
print('Number of Communities : ', len(np.unique(values)))
```

```
Number of Communities :  28
```

```
# Create Dataframe with Product Description and Community ID
products_communities = pd.DataFrame(nodes_labels, columns = ['product_description'])
products_communities['community_id'] = values
```

```
# Community 1
products_communities[products_communities['community_id'] == 1].head(40)
```

	product_description	community_id
1	MINI FUNKY DESIGN TAPES	1
47	BLUE ROSE FABRIC MIRROR	1
49	SET/9 CHRISTMAS T-LIGHTS SCENTED	1
102	SET/10 RED POLKADOT PARTY CANDLES	1
160	SKULLS WRITING SET	1
161	BALLOONS WRITING SET	1
162	DINOSAURS WRITING SET	1
281	COSY SLIPPER SHOES SMALL RED	1
308	HEART T-LIGHT HOLDER	1
309	STAR T-LIGHT HOLDER	1
310	CHRISTMAS TREE T-LIGHT HOLDER	1
372	VINTAGE PAISLEY STATIONERY SET	1
394	RIBBON REEL STRIPES DESIGN	1
490	HEART FILIGREE DOVE SMALL	1
491	HEART FILIGREE DOVE LARGE	1
497	OFFICE MUG WARMER POLKADOT	1
509	RETROSPOT PARTY BAG + STICKER SET	1
535	JUMBO BAG PINK POLKADOT	1
597	SET OF 2 TINS JARDIN DE PROVENCE	1
644	WOODEN HEART CHRISTMAS SCANDINAVIAN	1
652	PACK OF 6 BIRDY GIFT TAGS	1
674	SET OF 6 SOLDIER SKITTLES	1
708	JUMBO BAG DOLLY GIRL DESIGN	1
743	RIBBON REEL SOCKS AND MITTENS	1
745	RIBBON REEL CHRISTMAS PRESENT	1
749	MAKE YOUR OWN PLAYTIME CARD KIT	1
751	MAKE YOUR OWN MONSOON CARD KIT	1
761	SET OF 3 NOTEBOOKS IN PARCEL	1
796	SET OF 6 T-LIGHTS SANTA	1
797	SET OF 6 T-LIGHTS SNOWMEN	1
799	PACK 3 BOXES BIRD PANNETONE	1
843	PAPER CHAIN KIT VINTAGE CHRISTMAS	1
871	60 CAKE CASES VINTAGE CHRISTMAS	1
875	JAM MAKING SET WITH JARS	1
890	SOLDIERS EGG CUP	1
922	CHRISTMAS DECOUPAGE CANDLE	1
957	ASSORTED COLOUR BIRD ORNAMENT	1
973	THREE CANVAS LUGGAGE TAGS	1
989	SWIRLY CIRCULAR RUBBERS IN BAG	1
1017	DOORMAT BLACK FLOCK	1

Product Recommendation

This is the Phase where the Machine Learning Model is Trained to provide the Shopkeepers with Product Recommendations

```
# Turn into Dataframe
products_weighted_pd = pd.DataFrame(products_weighted, columns = nodes_labels)
products_weighted_pd.set_index(products_weighted_pd.columns, 'product', inplace=True)

products_prob = products_weighted_pd.divide(products_weighted_pd.max(axis = 1), axis = 0)
```

```
<ipython-input-25-792074395a0f>:3: FutureWarning: In a future version of pandas all arguments of DataFrame.set_index except for the argument 'keys
  products_weighted_pd.set_index(products_weighted_pd.columns, 'product', inplace=True)
```

```
# Saving the Trained Machine Learning Model for Use in the UI
products_prob.to_csv('products_prob.csv')
```

Testing

Let's now Test the Machine Learning Model (i.e. Product Recommendation System) if it throws proper Recommendations to the Shopkeeper

```
# Add Item to the Basket
basket = ['HOME BUILDING BLOCK WORD']

# Select the Number of Relevant Items to Suggest
no_of_suggestions = 3

all_of_basket = products_prob[basket]
all_of_basket = all_of_basket.sort_values(by = basket, ascending=False)
suggestions_to_customer = list(all_of_basket.index[:no_of_suggestions])

print('You May Consider Restocking : ', suggestions_to_customer)
```

```
You May Consider Restocking :  ['LOVE BUILDING BLOCK WORD', 'BATH BUILDING BLOCK WORD', 'ASSORTED COLOUR BIRD ORNAMENT']
```

✓ 0s completed at 6:54 PM